Free and Open Source Software for Geospatial (FOSS4G) Conference Proceedings

Volume 14 *Portland, Oregon, USA* Article 5

2014

Evaluation of Web Processing Service Frameworks

M. Ebrahim Poorazizi
University of Calgary (Canada)

Andrew J.S. Hunter

Evaluation of Web Processing Service Frameworks

by M. Ebrahim Poorazizi and Andrew J.S. Hunter

University of Calgary (Canada). mepooraz@ucalgary.ca

Abstract

As geoprocessing on the web has matured in recent years, an increasing number of geoprocessing services and functionality are becoming available in the form of online Web Processing Services (WPS). Consequently, the quality of such geoprocessing services is of importance to ensure that WPS instances fulfill users' expectations. In this paper, we illustrate, and discuss initial results from a quantitative analysis of the performance of WPS servers. To do so, we used two test scenarios to measure response time, response size, throughput, and failure rate of five WPS servers including 52° North, Deegree, GeoServer, PyWPS, and Zoo. We also assess each WPS server in terms of qualitative metrics such as software architecture, perceived ease of use, flexibility of deployment, and quality of documentation. A case study addressing accessibility assessment is used to evaluate the relative advantages and disadvantages of each implementation, and point to challenges experienced while working with these WPS servers.

Keywords: OGC WPS; Geoprocessing; Performance Evaluation; Benchmark.

1 Introduction

With the development of geospatial services, web-based GIS (Geographic Information Systems) have progressed towards a service-oriented paradigm (Mayer, Stollberg, & Zipf, 2009). Today, spatial services can be used to effectively support common tasks undertaken by spatial information users, for example, discovery and access to, process of, or visualization of spatial data. Catalogue Services for the Web (CSW), Web Feature Services (WFS), Web Coverage Services (WCS), Web Mapping Services (WMS), and WPS are common services defined by the OWS (Open Geospatial Consortium Web Service) initiative. A CSW provides the ability to publish and search collections of descriptive information (metadata) (Solntseff & Yezerski, 1974) for spatial data and services (Nebert, Whiteside, & Vretanos, 2007). A WFS is the main geospatial service for publishing vector spatial data, generally encoded using Geography Markup Language (GML) (Vretanos, 2002). A WCS defines a standard interface and operations that enable interoperable access to spatial coverage (Spatial information representing space/time-varying phenomena) datasets (Evans, 2003). A WMS delivers visualizations of data and, unlike WFS and WCS, does not deliver the data directly (de La Beaujardiere, 2004).

In the context of processing services, the Open Geospatial Consortium (OGC) has standardized the WPS interface for publishing of spatial processes, the discovery of, and binding to, those processes by users (Schut, 2007). A spatial process may include algorithms, calculations, or various kinds of models, which are exposed as a service instance, and operating on spatial data. A WPS, thus, can be used to design and develop a wide variety of GIS functionalities, and be made available to users across a network, as well as provide access to previously defined functions, calculations, or computational models.

With the emergence of geoprocessing on the web, the WPS specification and its (application) profiles have been applied to a wide array of use cases, from accessibility assessment (Steiniger, Poorazizi, & Hunter, 2013) to ecological modeling (Dubois, Schulz, Skøien, Bastin, & Peedell, 2013). The increasing use of WPS instances has also raised pertinent quality concerns — users/developers are likely to be concerned about the Quality of Service (QoS) attributes such as performance, reliability, and security.

The performance of a particular WPS is often of importance to users, arguably the most important, when evaluating the QoS of a specific service. Moreover, performance has a direct effect on other QoS attributes; for example, poor performance will affect reliability, scalability, capacity, accuracy, accessibility, and availability (Cibulka, 2013).

A developer's concerns, during designing and development of a WPS, are often twofold. As noted, from a quantitative perspective, performance is one of the key principles that can ensure both user and application developer satisfaction. From a qualitative point of view, quality concerns such as software architecture, perceived ease of use, flexibility of deployment, quality of documentation, and support accessibility are important factors that can guide developers during selection of a WPS framework that fits

a particular application domain best.

Several reviews have been reported in the literature that evaluates spatial services from both a quantitative and qualitative perspectives. MapServer's WMS has been assessed and optimized by Kalbere (2010). COSMC (Czech Office for Surveying, Mapping and Cadastre) and CENIA (Czech Environmental Information Agency) WMSs have been tested for availability and performance (Horák, Ardielli, & Horáková, 2009). Bermudez et al. (2009) compared the ability of WFS and SOS (Sensor Observation Service) to publish time series data. Tamayo et al. (2011) presented an empirical study of instances of servers implementing SOS in terms of compliance with OGC's SWE (Sensor Web Enablement) and interoperability, and in our previous work we evaluated performance of three SOS servers – 52° North, MapServer, and Deegree – based on different test scenarios (Poorazizi, Liang, & Hunter, 2012). Moreover, a WMS performance shootout has been presented annually since 2007 at the FOSS4G (Free and Open Source Software for Geospatial) conference, which provides a standardized procedure for measuring and comparing the performance of WMS server installations (http://wiki.osgeo.org/wiki/FOSS4G_Benchmark).

Within the geoprocessing domain, there have been few attempts to evaluate WPS servers. Scholten et al. (2006) investigated efficiency of web services for geoprocessing in a Spatial Data Infrastructure (SDI), but focused on caching, network adaptation, data granularity, and communication modes. Michaelis and Ames (2009) evaluated the WPS 0.4.0 specification, identified challenges, and proposed potential enhancements from an implementation perspective. In addition, a WPS shootout was presented at the FOSS4G conference 2011, which evaluated five WPS servers, 52° North, Deegree, GeoServer, PyWPS, and Zoo, in terms of compliance with OGC's WPS, and interoperability (http://wiki.osgeo.org/wiki/WPS_Shootout). The main achievement of the aforementioned works is that they concentrated on influential performance issues, the WPS protocol and its specification, and compliance and interoperability testing. However, there is also a need to evaluate WPS functionality and performance. Through performance evaluation, WPS developers can (i) identify the strengths and weaknesses of each system, and (ii) improve WPS servers to meet both application user and developer requirements (Zhu, 2003). These issues are addressed in this paper. We have evaluated the performance of five WPS servers – 52° North, Deegree, GeoServer, PyWPS, and Zoo – using two test plans in an accessibility assessment scenario. To do so, the WalkYourPlace Transit Model (Steiniger et al., 2013) was used to design the geoprocessing workflow. The workflow was then developed using Python and wrapped and exposed as a standard WPS using the candidate WPS servers. The sample locations were selected using a stratified random sampling approach within the bounds of the City of Calgary, Alberta, Canada. During experiments we controlled the number of concurrent requests, and the WPS input parameters to assess the performance and load capacity of the WPS servers. The remainder of the paper is structured as follows. Section two introduces the WPS specification. The specification of candidate WPS servers is described in section three. Section four explains the methodology used to evaluate the WPS servers, along with a description of the case study, technical architecture, test scenarios, and hardware configuration of the servers used. Section five presents the result. In section six, the WPS servers are assessed in terms of qualitative metrics. Section seven summarizes our findings.

2 Web Processing Service

The OGC released version 1.0.0 of the WPS specification in June 2007 (Schut, 2007). The specification, along with the OGC Web Processing Service Best Practice discussion paper, describe a web service interface that defines how a client and server should cooperate during the execution of a spatial analysis, and how results of the process should be presented (Schäffer, 2012). Clients can send requests via three core operations using three methods: Key Value Pairs (KVP) encoding via HTTP's (HyperText Transfer Protocol) GET, XML (eXtensible Markup Language) via HTTP's POST, or a SOAP/WSDL (Simple Object Access Protocol/Web Service Description Language) approach. The WPS specification defines three mandatory operations that enable spatial processing on the Internet (Schut, 2007). The GetCapabilities operation allows a client to request and receive service metadata documents that describe the capabilities of a specific server implementation. The DescribeProcess operation returns detailed information about a process' requirements, such as input and output parameters, as well as allowable data formats. The Execute operation invokes a specific process implemented by the WPS, using the input parameters provided, and returns the results of the service to a client.

3 WPS Servers

In this paper five WPS servers were used for performance evaluation. 52° North WPS (http://52north.org/communities/geoprocessing/wps/) is developed by the 52° North Initiative for Geospatial Open Source Software GmbH. It implements the three mandatory operations of the WPS 1.0.0 specification. The 52° North WPS server is realized as a servlet and can be deployed in any servlet container such as Apache Tomcat (http://tomcat.apache.org/). Developing a custom WPS process is implemented using 52° North's WPS SDK (Software Development Kit) to define parameters necessary for service configuration, service metadata, and business logic. Spatial analysis functions can be integrated using various libraries such as JTS (http://www.vividsolutions.com/jts/JTSHome.htm), GeoTools (http://www.geotools.org/), R (http://www.r-project.org/), GRASS (http://grass.osgeo.org/), SEXTANTE (http://www.sextantegis.com/), and ArcGIS Server (http://www.esri.com/software/arcgis/arcgisserver), for example.

Deegree WPS (http://www.deegree.org/) is a service built into the Deegree Java framework for geospatial applications and OGC service implementations, deegree 3. deegree 3 is an Open Source Geospatial (OSGeo) Foundation project. It supports the core profile operations of the WPS 1.0.0 standard specification. The Deegree WPS server is implemented as a servlet and can be deployed in any servlet container, i.e., Apache Tomcat. Developing a custom process requires the creation of a Maven (http://maven.apache.org/) project. Configuration parameters and service metadata are defined through XML configuration files and business logic is implemented as a Java class. Deegree WPS currently supports the SEXTANTE spatial library, but other spatial libraries such as FME (http://www.safe.com/fme/fme-technology/) and GRASS (http://grass.osgeo.org/) are being considered.

GeoServer WPS (http://docs.geoserver.org/wps) is part of the popular open-source GIS project GeoServer, a project of the OSGeo Foundation. It supports the three mandatory operations contained in the WPS 1.0.0 specification. The GeoServer WPS server is built using Java technology as a servlet, and runs in an integrated Jetty or Apache Tomcat web server environment. Developing a custom process is accomplished by creating a Maven (https://maven.apache.org/) project. Configuration parameters and service metadata are defined through XML configuration files, and business logic is implemented as a Java class. GeoServer WPS supports GeoTools and JTS spatial libraries.

PyWPS (http://pywps.wald.intevation.org/) is a Python-based WPS implementation developed by Intevation GmbH. It implements the mandatory operations of the WPS 1.0.0 specification. It runs as a CGI (Common Gateway Interface) application and can therefore be deployed in any HTTP Server environment, Apache HTTP Server, for example. Developing a custom process requires the creation a python file to implement the business logic and define service metadata and configuration parameters. PyWPS enables access to a wide range of analysis functions via GRASS, GDAL (http://www.gdal.org/), and R libraries.

Zoo (http://www.zoo-project.org/ is an OSGeo Foundation project that enables existing open source libraries to interact through its WPS framework. It supports the mandatory operations of the WPS 1.0.0 specification. It runs as a CGI application and so can be deployed in any HTTP Server environment. Developing a custom process requires the creation of a configuration file (.zcfg) that defines service metadata and configuration parameters. Business logic can be implemented in several programming languages including C/C++, PHP, JavaScript, Java, Perl, Python, or FORTRAN. Several spatial libraries such as GRASS, GEOS (http://trac.osgeo.org/geos/), and GDAL are supported by default in Zoo WPS framework.

Table 1 lists the technical characteristics of 52° North, Deegree, GeoServer, PyWPS, and Zoo WPS servers.

4 Methodology

In this section, we explain the methodology used to test and measure the performance of the WPS servers.

4.1 Case Study

In order to evaluate performance of the WPS servers, we used the WalkYourPlace Transit Model (http://webmapping.ucalgary.ca/WPSClient/), which is one of the accessibility assessment models developed for the PlanYourPlace project (Steiniger et al., 2013). Based on this model, if the users provide (i) their current location, or perhaps a location they would like to start walking from, (ii) a maximum time they are

	52°North	Deegree	GeoServer	PyWPS	Zoo
Development Platform	Java	Java	Java	Python	C/C++
License	GNU GPL v2	LPGL	GNU GPL v2	GNU GPL v2	MIT/X-11 style
Supported Libraries	JTS GeoTools SEXTANTE R GRASS ArcGIS	SEXTANTE	JTS GeoTools	GRASS GDAL R	GRASS GEOS GDAL
Natively Supported Languages for Process Development	Java	Java	Java	Python	C/C++ Fortran Java Python PHP Perl JavaScript
Service	Servlet	Servlet	Servlet	CGI	CGI
DCP Request	GET, POST, SOAP	GET, POST, SOAP	GET, POST	GET, POST, SOAP	GET, POST

Table 1: WPS servers' technical specifications.

willing to walk to a point of interest, or a transit stop, (iii) average walk-speed, (iv) a maximum time they would like to wait for transit, and (v) and the maximum time they would like to travel by transit, then the system will evaluate the extent of the area that is accessible using pedestrian and transit infrastructure. The services within an accessibility area are then analysed (e.g., point of interests (POI) such as parks, stores, libraries, etc.) to determine an accessibility score for the accessibility area. Should the user wish, they can ask for a distance decay function to be applied that discounts the contribution of POIs that are further away from the users start location. Next, an assessment of crime is undertaken for the accessibility area. The accessibility area, accessibility score, and the crime index are final outputs of the model. For more details about accessibility assessment models deployed as part of the WalkYourPlace framework see Steiniger et al. (2013).

4.2 Technical Architecture

Figure 1 illustrates the processing service architecture for the WalkYourPlace Transit Model. The service architecture has been designed to reduce complexity and enable reuse of geoprocessing services. From a service design perspective, a bottom-up (Granell, Díaz, & Gould, 2010) approach was used to design the services. The geoprocessing services were then implemented using Python in such a way that to be accessible via HTTP GET/POST. In this context, PostGIS spatial functions were used to perform geometric computations such as calculating distances between pairs of points, calculating the area of polygons, and merging multiple geometric objects. Remaining functionality was developed using Python libraries. The geoprocessing services were then wrapped and exposed as standard WPSs using 52° North, Deegree, GeoServer, PyWPS, and Zoo frameworks. In this context, the WPS server acts as a gateway, which enables standard communication with the back-end (Python-based) geoprocessing services. It actually accepts the Execute request, parses the query, and sends it to the corresponding Python-based service using HTTP handlers. After getting the result, the WPS server prepares it as a standard WPS response and sends it back to the client. In this study, we developed seven Python-based geoprocessing modules to perform the analysis, and seven WPS instances using each WPS server to wrap and expose them as standard WPS services (see Figure 1). A PostgreSQL/-PostGIS database was used to store various spatial datasets such as the street and transit networks, the transit schedule, and crime data, obtained originally from OpenStreetMap, Calgary Transit, and the Calgary Police, respectively. To search for attractions within accessibility areas, POI datasets were fetched on demand from OpenStreetMap and MapQuest databases using REST (REpresentational State Trans-

fer) APIs (Application Programming Interfaces). For the calculation of transit-based accessibility areas we used the General Transit Feed Specification (GTFS) formatted data published by the City of Calgary.

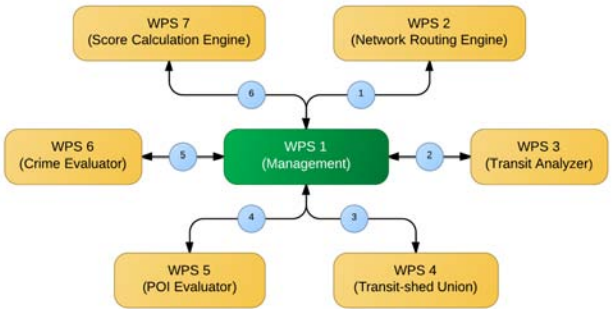

Figure 1. The WalkYourPlace processing service architecture.

The geoprocessing service framework includes an accessibility assessment engine that performs the accessibility analysis through chaining of geoprocessing services in a multi-step pattern, i.e. a workflow. To achieve desired application flexibility, service reusability, and improve performance, the workflow-managed chaining method was used (Alameh, 2003).

Figure 2 presents a UML sequence diagram that outlines how an accessibility score is calculated for pedestrian and transit infrastructure. The client sends a WPS Execute command to the Management WPS, which then initiates an Execute call to the Walkshed WPS. The Walkshed WPS returns a GeoJSON polygon of the network-based accessibility area. The Management WPS then sends an Execute request to the Transit WPS to find all transit stops within the accessibility area and generates an accessibility area for each transit stop based on the user defined constraints described in section 4.1. The Transit WPS returns a GeoJSON-encoded multi-polygon feature. Next, the Management WPS sends an Execute request to the Union WPS to merge all the accessibility areas generated by the Transit WPS. The Union WPS returns a single polygon feature encoded as GeoJSON. The Management WPS then sends an Execute request to the POI WPS to find all attractors within the accessibility area. The POI WPS returns a point set of services encoded as GeoJSON points, along with attributes describing the types of features found. The Management WPS then repeats the same request to the Crime WPS to obtain incident locations. Finally, the Management WPS sends an Execute request to the Aggregation WPS along with the accessibility polygon, the responses from the POI and Crime WPS's, and a Boolean variable to indicate whether the distance decay function should be applied or not. The response from the Aggregation WPS includes an accessibility score, a crime score, and an accessibility area. The Management WPS then returns the Aggregation WPS's response to the client for presentation.

4.3 Test Scenario

In this study, to ensure the same test conditions for all WPS servers were used, we developed the geoprocessing services using Python and then wrapped and exposed them as WPS services. Given this implementation the WPS servers (i.e., 52° North, Deegree, GeoServer, PyWPS, and Zoo) act as a gateway that enables standard interaction between clients (i.e., the user or other services) and back-end geoprocessing services, which implemented using Python. For example, when the client sends an Execute request to the Management WPS, it then sends a request to a corresponding Python service, which is accessible via HTTP GET/POST. After getting the response, the Management WPS sends Execute requests to other WPS services (i.e., Walkshed, Transit, Union, POI, Crime, and Aggregation WPSs), which in turn communicate with back-end Python services to get the processing result. As such, the Execute method depends on external service calls, and the response time for invocation of the whole workflow (Figure 2) was measured to evaluate the "end-to-end" performance, i.e., the response time includes communication time and processing time.

To evaluate the performance of the WPS servers, we designed two test scenarios based on the accessibility assessment case study. In the first scenario (Scenario A), we randomly chose the WPS input parameters to generate 45 Execute requests. The number of concurrent requests was assumed constant (n=1). The input parameters were selected using the following criteria:

- Walking Start Point: sample locations were selected using a stratified random sampling approach within the bounds of the City of Calgary (see Figure 3).
- Walking Start Time: random timestamps between 5 a.m. and 12 p.m., which is the Calgary Transit hours of operation (http://www.calgarytransit.com/accesscalgary/hours.html).
- Walking Time Period: we selected random values between 5 minutes and 20 minutes.

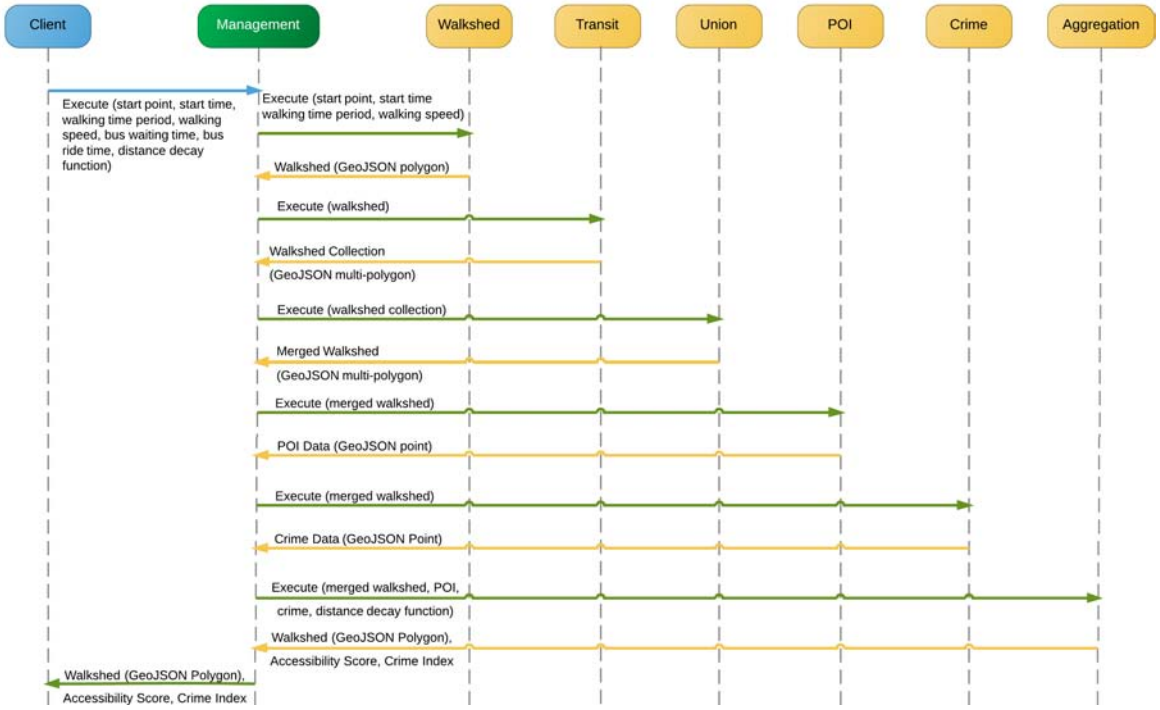

Figure 2. UML activity diagram of accessibility assessment workflow.

- Walking Speed: we selected random values between 3 km/h and 6 km/h with step values of 0.5 km/h.
- Bus Waiting Time: we selected random values between 0 minutes and the Walking Time Period.
- Bus Ride Time: we selected random values between 0 minutes and Walking Time Period – Bus Waiting Time.
- Distance Decay Function: a Boolean variable (i.e., True/False) was selected randomly.

For the second scenario (Scenario B), we focused on the number of concurrent requests. In this context, the number of concurrent requests was generated using a 2^n pattern, while variable "n" was selected between 0 and 7 with step value of 1. 30 WPS Execute requests were generated for each WPS service and replicated according to the concurrent request pattern. All other criteria were determined using the above mentioned approach for Scenario A.

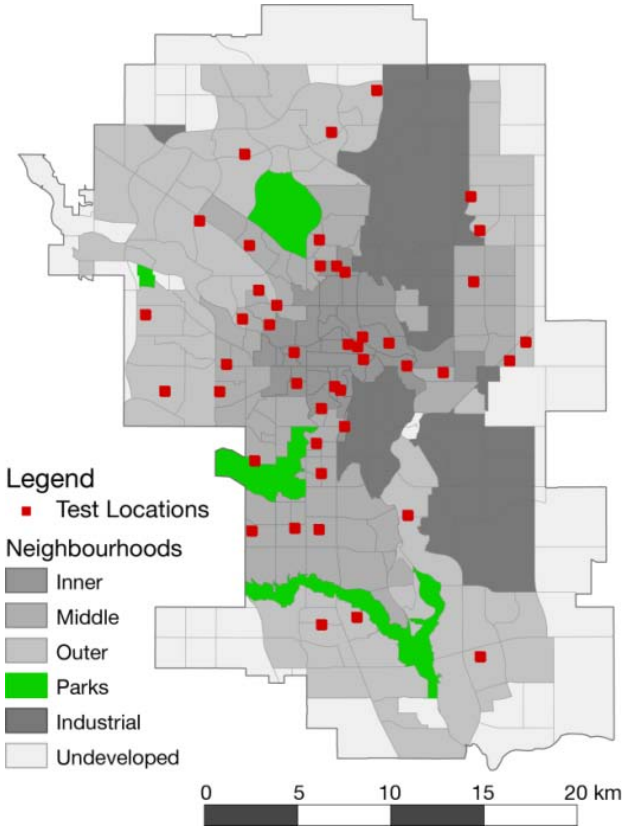

Figure 3. Map of the City of Calgary highlighting the locations used for evaluating the WPS servers.

4.4 Test Environment

To more accurately reflect the users experience, all the tests have been measured from the client-side. On the server-side, a Dell OptiPlex 990 was used as the host machine, with an Intel Core i5 (3.1GHz) CPU, 8GB of RAM, and 500GB of disk space, running Microsoft Windows 7 Professional (64-bit). In order to deploy and test the WPS servers under the same conditions, each WPS package was installed on a separate virtual machine with the same hardware configuration. VMware Player 5.0.1 (http://www.vmware.com/) was used to setup five virtual machines with access to 4GB of RAM, 40GB of disk space, and use of 4 out of 8 CPU cores, running Ubuntu 12.04 LTS (64-bit). The network configuration of the virtual machines was set to "Bridged", allowing them to connect directly to the physical network and obtain a dedicated IP address. Table 2 summarizes the configuration of the server machine (host), and virtual machines. For more information about the configuration of database server and software libraries used see Appendix A.

Hardware	**Dell OptiPlex 990 (Host)**	**VMware (VM)**
CPU	Intel Core i5 3.1GHz	4 Cores of 8
RAM	8GB	4GB
HDD	500GB	40GB
OS	Windows 7 Professional (64-bit)	Ubuntu 12.04 LTS (64-bit)

Table 2. Experimental server configuration.

The machine used to run the tests at the client-side was the host machine. In this study, we used the same machine to set up the servers and test them, while according to (VMware, 2006), "an ideal setup for workloads that involve network traffic is to use an external client (on a different physical system) to send network traffic to and receive network traffic from a virtual machine". Although this could affect the performance of the WPS servers, the test conditions (i.e., hardware and software configurations) were the same for all the servers, which are shown in Table 2, Table 6, and Table 7. It was assumed that network time would be constant and therefore would not contribute significantly to differences in response times.

In order to run the tests and measure performance factors (e.g., response time, response size, etc.), Apache JMeter (http://jmeter.apache.org/) was used, as it is a widely accepted performance-testing tool for web applications.

5 Experimental Results

Since each WPS server uses database connections to execute queries, a warm-up run was first performed. This ensures that the overhead of establishing a connection to the database is not accounted for in the metrics (elapsed time). During each performance test, only one virtual machine was run. Response time, response size, and whether or not a request was successful were logged. This data allowed the estimation of average response times, average server throughput, average server failure rate, and average response size returned by each WPS server. Figure 4 to Figure 7 and Table 3 below report the results of the experiments.

First, the performance test for Scenario A is reported. The average response time, time taken for a service call to return all response bytes, the average response size, the quantity of data exchanged between client and server, for each of the WPS servers are listed in Table 3 and plotted on Figure 4.

Given the data, the most rapid WPS server was Deegree, with an average response time of 2.499 $\pm$ 1.259 s (95% confidence interval (CI)), followed by GeoServer WPS, 52° North WPS, Zoo WPS, and PyWPS. A one-way Analysis of Variance (ANOVA) test indicates that all WPS servers respond similarly with no significant difference between them, $F(4,220)=0.739$, $p=0.566$.

In terms of response size, there was no significant difference ($F(4,220)=1.071$, $p=0.372$) either with all WPS servers returning similar response package data volumes ($\approx$ 2.484 kB). The GeoServer WPS returned the least amount of data (2.301 $\pm$ 0.267 kB) to the client, and PyWPS had the most (2.686 $\pm$ 0.269 kB).

The reason of having different response sizes was because of a slight different in XML tags within the Execute response. For example, wps:ExecuteResponse content is listed in Table 4 for PyWPS and GeoServer WPSs' Execute response, which returned the most, and the least amount of data, respectively.

Scenario B was designed to assess the effect of increased load on each server. The effect of load was assessed by increasing the number of concurrent requests from 1, to 2, 4, 8, 16, 32, 64, and finishing with 128 concurrent requests. Individual services and the service chain were tested using pre-defined input pa-

rameters under normal condition (n=1) and no error was observed. Those parameters were then used to measure the performance of the WPS servers under high loads (n>1). To get representative results, all of the experiments were repeated 30 times and the response time, response size, and server success/failure were recorded. These data allowed the estimation and comparison of server throughput. The results are depicted in Figure 5 to Figure 7.

WPS Server	Response Time (s)	Response Size (kB)
52° North	2.784 ± 1.269	2.448 ± 0.267
Deegree	2.499 ± 1.259	2.505 ± 0.267
GeoServer	2.753 ± 1.255	2.301 ± 0.267
PyWPS	3.995 ± 1.661	2.686 ± 0.269
Zoo	2.999 ± 1.313	2.479 ± 0.269

Table 3. Results for Execute request (Scenario A).

Figure 5 shows that Deegree, GeoServer, and Zoo generally perform similarly, the only difference is an improvement in response time by Deegree for 128 concurrent requests. With an increase from 64 to 128 concurrent requests Deegree's response time improves to approximately half that of PyWPS and Zoo. It is apparent that 52° North and PyWPS had difficulty when more than 16 and 64 concurrent requests were received for processing respectively. It is also evident that when more than 64 concurrent requests were sent to PyWPS and Zoo WPS servers failure rates increased dramatically, approaching 100% at 128 concurrent requests. Throughput was also affected significantly by the number of concurrent requests, particularly for 52° North, which returned less than 1 successful request per hour once concurrent requests increased above 16. All servers performed substantially better when only one request was received at a time, with 52° North achieving a throughput of 1,445 successful requests per hour, followed by Zoo with 1,145, GeoServer with 1,115, Deegree with 1,024, then PyWPS with 894 requests per hour.

Because of the variation in the data, when analyzing the results using a two-way ANOVA, only the number of concurrent requests had an effect on load testing ($F(1,30)=20.640$, $p<0.001$), individual servers did not contribute to differences observed. As the number of concurrent requests increased GeoServer and Zoo followed a similar (linear) trend. Deegree tended to perform better, especially under high loads (n=128). In addition, 52° North and PyWPS failed to respond while processing more than 16 and 64 requests respectively. The failure rate of Deegree and GeoServer exhibited a same pattern. We observed a failure rate of 0.8% under high loads (n > 4). PyWPS and Zoo also followed a same failure rate pattern. It was constant (≈1.6%) between four and 64 concurrent requests and then approached 100% under higher loads (n > 64). All the WPS servers performed similarly in terms of throughput, for example with four concurrent requests they processed around 600 requests per hour. It suggests that the WPS servers were capable of handling a request every six seconds (n = 4). This result requires further investigation to determine if the servers can be tuned to function more effectively under real-world conditions. These results are summarized in Figure 5 to Figure 7.

6 Lessons Learned

In this section, the relative advantages and disadvantages of each WPS server, and challenges experienced while working with them are discussed. In this context, the WPS servers were evaluated from a qualitative perspective in terms of: ease of installation and configuration; perceived ease of use and flexibility for creating new processes; native support for development languages; quality of documentation; and community support. The qualitative comparison results are shown in Table 5.

Installation – as 52° North WPS, Deegree WPS, and GeoServer WPS servers are servlet-based applications, the installation process was straightforward. For 52° North and Deegree, installation is accomplished by deploying the downloaded/built WAR (Web ARchive) file into a servlet container such as Apache Tomcat. For GeoServer, after deploying the WAR file into a servlet container, the WPS Extension should be extracted to the WEB-INF/lib directory of the GeoServer installation. Library dependency was the main issue with PyWPS and Zoo WPS servers' installation process. They have several library dependencies that must be installed first. PyWPS follows a typical Python installation procedure using a setup.py script. Further configuration is necessary to set server paths, and the process folder locations. Installation of the Zoo Kernel, configuration, and installation of the Zoo Service Provider were the main steps required to deploy a service on the Zoo WPS server.

Creating a new process and configuration – as 52° North WPS, GeoServer WPS, and Zoo WPS

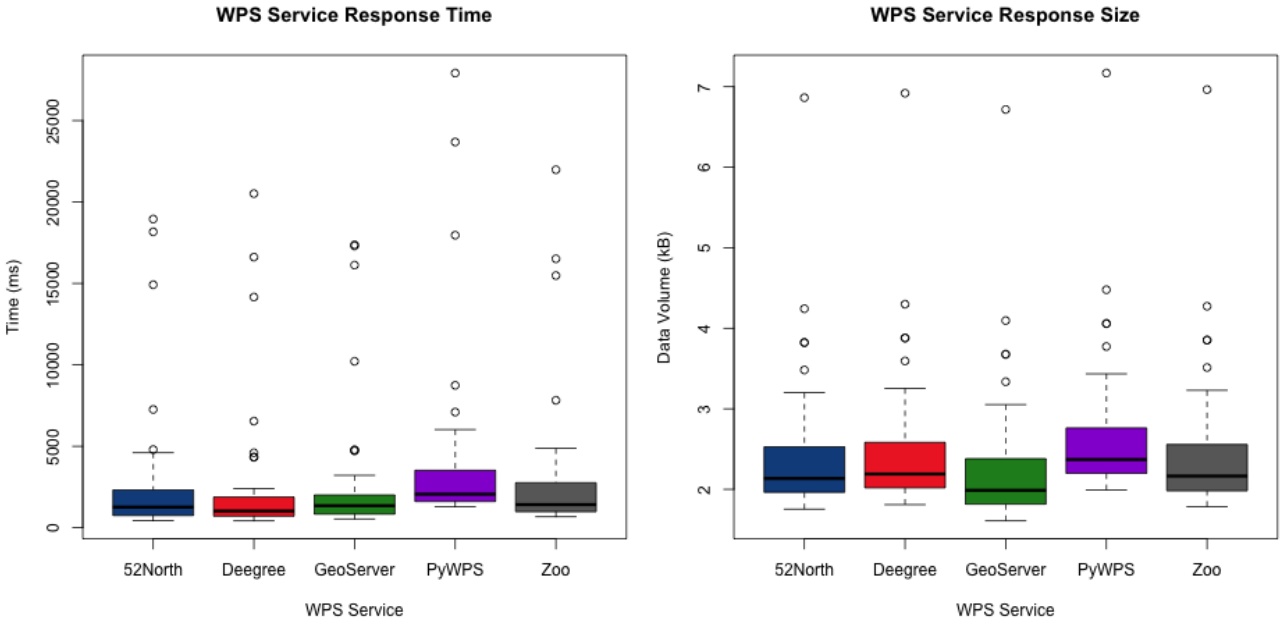

Figure 4. Response time (left) and size (right) for Execute requests (Scenario A).

WPS Server	The Execute Response
PyWPS	<wps:ExecuteResponse xmlns:wps="http://www.opengis.net/wps/1.0.0" xmlns:ows="http://www.opengis.net/ows/1.1" xmlns:xlink="http://www.w3.org/1999/xlink" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.opengis.net/wps/1.0.0 http://schemas.opengis.net/wps/1.0.0/wpsExecute_response.xsd" service="WPS" version="1.0.0" xml:lang="en-CA" serviceInstance="http://localhost/cgi-bin/wps? service=WPS&request=GetCapabilities&version=1.0.0" statusLocation="http://localhost/wpsoutputs/pywps-137824772530.xml">
GeoServer	<wps:ExecuteResponse xml:lang="en" service="WPS" serviceInstance="http://10.146.29.24:8080/geoserver/ows?" version="1.0.0" xmlns:wps="http://www.opengis.net/wps/1.0.0" xmlns:ows="http://www.opengis.net/ows/1.1" xmlns:xlink="http://www.w3.org/1999/xlink">

Table 4: A portion of the Execute response document returned by PyWPS and GeoServer WPS.

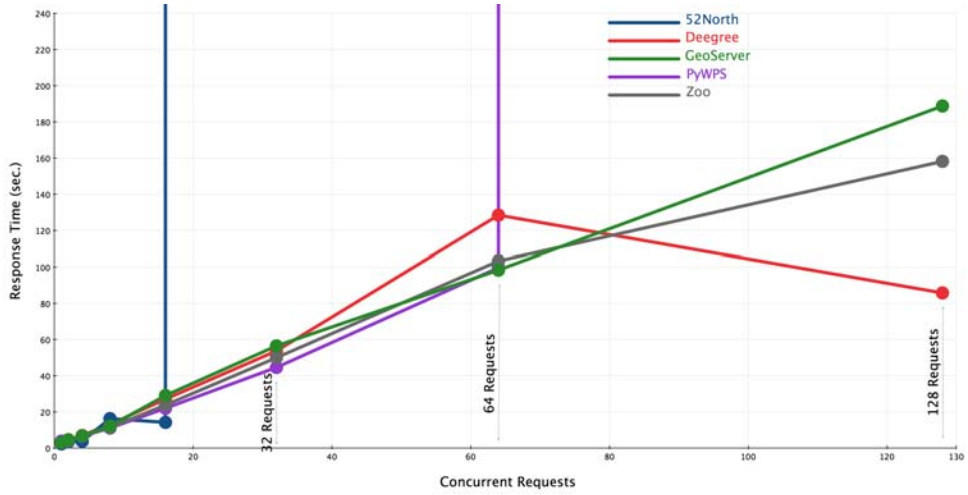

Figure 5. Response time when increasing concurrent requests.

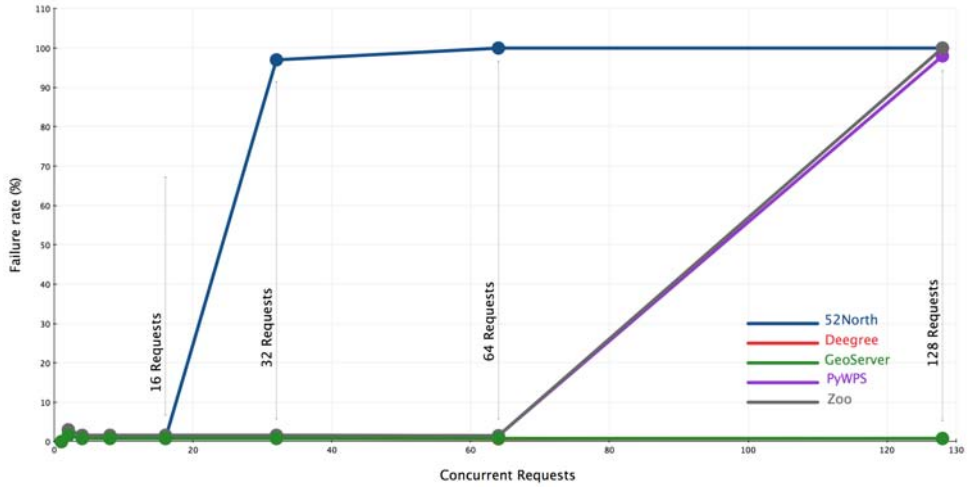

Figure 6. Failure rate with increasing concurrent requests.

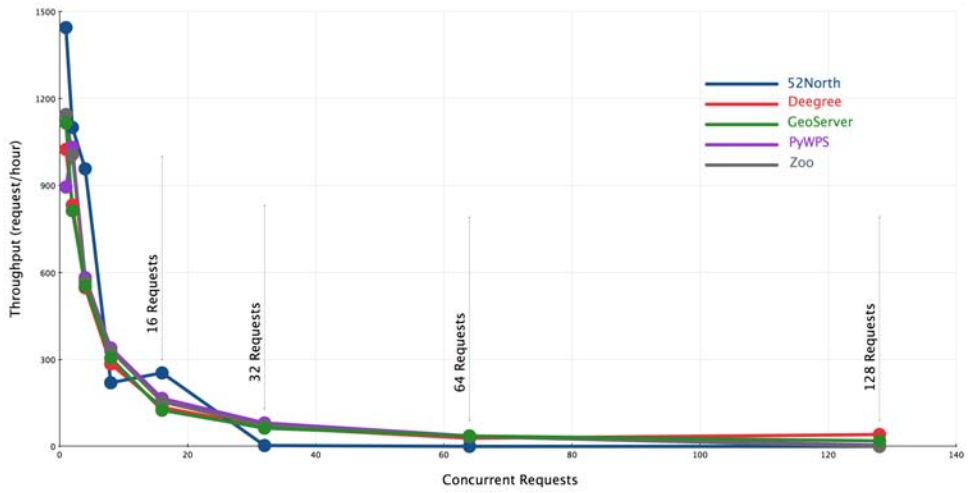

Figure 7. Throughput with increasing concurrent request.

	52°North	Deegree	GeoServer	PyWPS	Zoo
Installation*	Easy	Easy	Easy	Difficult	Difficult
Create new processes and Configuration*	Easy	Medium	Easy	Difficult	Easy
Native Development Languages	Java	Java	Java	Python	C/C++ Fortran Java Python PHP Perl JavaScript
Quality of Documentation**	Great	Good	Great	Good	Good
Community Support	Mailing list, Wiki, Forum, Issue Tracker, SVN, GitHub	Mailing list, Wiki, Forum, Issue Tracker, SVN, GitHub	Mailing list, Forum, Issue Tracker, SVN, IRC Meeting, GitHub	Mailing list, GitHub	Mailing list, Forum, Issue Tracker, SVN, GitHub

* Ranking ranges: Easy; Medium; Difficult
** Ranking ranges: Weak; Good; Great

Table 5. WPS servers and their features from a qualitative perspective.

frameworks were well-documented, a new process was simple to create and easy to configure. For 52° North WPS, this procedure was accomplished using the WPS SDK in three steps: (i) create a Java class for the process, (ii) export the process as a JAR (Java ARchive) file, and (iii) deploy the process into 52° North's WPS framework. For GeoServer WPS, a new process is developed by creating a Maven project in three steps: (i) create a Java class and an XML configuration file for the process, (ii) compiling the project as a JAR file, and (iii) deploying the process into GeoServer's WPS framework. To create a new process for Zoo WPS, two steps have to be completed: (i) create a service file using one of the supported programming languages, and a zcfg configuration file for the process, and (ii) deploy the CGI application into Zoo's WPS framework. Although Deegree's documentation (http://download.deegree.org/documentation/3.3.3/html/), was well-organized and comprehensive, it was not particularly clear how to build and deploy a new process within Deegree's WPS framework, nor were there many examples to base development on. However, a Maven project should be created and three steps should be followed to add a new process to Deegree's WPS: (i) create a Java class and an XML configuration file for the process, (ii) compile the project as a WAR file, and (iii) deploy the servlet application into any servlet container. To add a new process to PyWPS framework, two steps should be followed: (i) create a service file and modify the configuration files (i.e., pywps.cfg and pywps.cgi), and (ii) deploy the CGI application into PyWPS's framework. On occasion the PyWPS server returned an HTTP Error 500 that prevented it from fulfilling WPS requests, especially after a new process had been added. To resolve this failure several access permission settings were required (for more details see Hamre (2011)).

Native Development languages – 52° North WPS, Deegree WPS, GeoServer WPS, and PyWPS frameworks support one native programming language each for the development of a new process, while the Zoo WPS framework supports seven programming languages. This adds flexibility for developers, as they are able to either develop new processing services in their language of choice, or develop services as independent modules that may draw on libraries from many different languages.

Quality of documentation – 52° North WPS and GeoServer WPS documentation was comprehensive

and provide clear instruction for installation and configuration of the WPS servers, along with clear instructions for developing new process instances.

Community support – 52° North WPS, Deegree WPS, GeoServer WPS, and Zoo WPS frameworks have large communities of users/developers and provide different communication mediums to support them. PyWPS does not appear to have an active community of users/developers, which may make access to support difficult.

7 Discussion and Conclusions

We have evaluated performance of WPS servers using two test scenarios via a case study that focuses on accessibility assessment. In the first scenario, the WPS servers were tested using 45 randomly generated Execute requests, holding the number of concurrent requests constant (n=1). The results show that on average Deegree returns the response package most rapidly. However, a one-way ANOVA test showed that, given the data, there is no significant difference in response time between the WPS servers tested ($F(4,220)=0.739$, $p=0.566$), nor data volume returned ($F(4,220)=1.071$, $p=0.372$).

In the second scenario, load testing was undertaken by varying the number of concurrent requests. Overall Deegree and GeoServer performed similarly, although Deegree tended to perform better under high loads. 52° North had difficulty when more than 16 concurrent requests were received for processing, but performed more effectively under low loads compared to other WPS servers. Under low loads, n=1, 52° North had the highest throughput completing 1,445 requests per hour, followed by Zoo with 1,145 requests per hour, GeoServer with 1,115 requests per hour, and Deegree and PyWPS completing 1,024 and 894 requests per hour respectively. Throughput for 52° North effectively went to zero requests per hour once the load increased to more than 16 concurrent requests. Although no failed requests were encountered under low loads, n=1, success rate for Deegree and GeoServer stabilized at four or more concurrent requests, to approximately 99.2%. PyWPS and Zoo followed the same pattern, with a success rate of 98.4% between four and 64 concurrent requests.

While four CPU cores were allocated to each WPS server during testing, upon reviewing CPU load logs it was evident that, except for PyWPS, only one CPU core was generally being used at any time during testing. Specifically, Deegree used only one CPU core; 52° North, GeoServer, and Zoo each used around 20% of one CPU core and 5% of the other cores. PyWPS used all cores during testing. In addition, memory usage of all WPS servers was constant (with minor fluctuations) during testing. On average memory use was 30%. This suggests that performance improvements may be possible if server specific tuning, or more effective development strategies are implemented. For example, the use of multiple CPU cores in Java-based applications is handled via JVM (Java Virtual Machine), which generally tends to be problematic. In this context, if particular implementation approaches or software libraries (e.g., concurrency libraries) are used it may result in a better performance.

We must also note that a WPS server's response time is dependent upon the intensity of a service's processing requirements. As such, performance results will depend on the complexity of the workflow, the complexity of individual back-end processes, and the complexity of the data.

The WPS servers have also been assessed in terms of qualitative metrics. 52° North WPS, Deegree WPS, and GeoServer WPS servers are easy-to-install and are well documented. They also have worldwide communities of developers/users, and provide different ways of communication to support their users/developers. The documentation for PyWPS was not complete, nor was it always clear and concise, making it difficult to install and configure the PyWPS server. PyWPS does not appear to have an active community of users/developers, and users/developers, as a consequence, may suffer from lack of support. Zoo WPS does have accessible documentation and an accessible support community. It also supports several programming languages and offers powerful and flexible approaches to develop WPS instances. Generally, compared to other WPS servers, 52° North and GeoServer seem to be the best choices when considering qualitative metrics, as they met most of the evaluation criteria we chose in this study.

It should be noted that standard compliance is a major issue in the WPS domain, which was not investigated in this study. Interoperability and standard compliance tests can be undertaken as a part of qualitative evaluation process, which focuses on schema, semantics, and encodings.

To conclude, when selecting an appropriate WPS server, we believe it is important to consider both quantitative and qualitative metrics. The importance of each metric can be weighted based on different application requirements. Generally speaking, from a user's perspective, performance is one

of the most important factors when choosing a web-based application, while from developers' perspective, qualitative factors such as perceived ease of installation and configuration, variety of development languages, quality of documentation and accessibility of support may be more critical. To choose a WPS server, we suggest starting an evaluation process with a basic set of questions that are linked to the evaluation criteria. The questions could be "who is the user of the system?" "What should the end-user be able to do with the system?" "What programming languages are developers comfortable with for develop of the system?" "How complex are the back-end processes?" "How should the system function, synchronous or asynchronous?" "What is the architecture used to design the processing workflow?" "What is the expected number of users?" In the end, the most appropriate WPS server should be selected based on a trade-off between quantitative performance metrics and qualitative "ease of use" metrics for a specific application or use case. This may lead to the selection of different WPS servers for different applications.

Note: the developed Python-based geoprocessing services, WPS instances, and test scripts are publicly available, see Appendix B.

References

Alameh, N. (2003). Chaining geographic information Web services. IEEE Internet Computing, 7(5), 22 – 29. doi:10.1109/MIC.2003.1232514

Bermudez, L., Cook, T., Forrest, D., Bogden, P., Galvarino, C., Bridger, E., … Graybeal, J. (2009). Web feature service (WFS) and sensor observation service (SOS) comparison to publish time series data. In Proceedings of International Symposium on Collaborative Technologies and Systems, 2009 (CTS '09) (pp. 36 –43). Baltimore, MD , USA. doi:10.1109/CTS.2009.5067460

Cibulka, D. (2013). Performance Testing of Web Map Services in three Dimensions – X, Y, Scale. Slovak Journal of Civil Engineering, XXI(1). doi:10.2478/sjce-2013-0005

De La Beaujardiere, J. (2004). Web Map Service (No. OGC 04-024). Open Geospatial Consortium Inc. Retrieved from portal.opengeospatial.org/files/?artifact_id=5316

Dubois, G., Schulz, M., Skøien, J., Bastin, L., & Peedell, S. (2013). eHabitat, a multi-purpose Web Processing Service for ecological modeling. Environmental Modelling & Software, 41, 123–133. doi:10.1016/j.envsoft.2012.11.005

Evans, J. D. (2003). Web Coverage Service (WCS), Version 1.0.0 (No. OGC 03-065r6). Open Geospatial Consortium Inc. Retrieved from portal.opengeospatial.org/files/?artifact_id=3837

Granell, C., Díaz, L., & Gould, M. (2010). Service-oriented applications for environmental models: Reusable geospatial services. Environmental Modelling & Software, 25(2), 182–198. doi:10.1016/j.envsoft.2009.08.005

Hamre, T. (2011, December 29). Open Service Network for Marine Environmental Data - WPS Cookbook. http://netmar.nersc.no/. Retrieved from http://netmar.nersc.no/sites/netmar.nersc.no/files/NETMAR_D7.7_WPS_Cookbook_r1_20111229.pdf

Horák, J., Ardielli, J., & Horáková, B. (2009). Testing of web map services. International Journal of Spatial Data Infrastructures Research,(Special Issue: GSDI 11), 25. Retrieved from http://www.gsdi.org/gsdiconf/gsdi11/papers/pdf/330.pdf

Kalbere, P. (2010). Performance and statistical analysis of WMS servers. In Proceedings of FOSS4G 2010. Barcelona, Spain.

Mayer, C., Stollberg, B., & Zipf, A. (2009). Providing Near Real-Time Traffic Information within Spatial Data Infrastructures. In Proceedings of International Conference on Advanced Geographic Information Systems Web Services (GEOWS '09) (pp. 104 –111). Cancun, Mexico. doi:10.1109/GEOWS.2009.17

Michaelis, C., & Ames, D. (2009). Evaluation and Implementation of the OGC Web Processing Service for Use in Client-Side GIS. GeoInformatica, 13(1), 109–120. doi:10.1007/s10707-008-0048-1

Nebert, D., Whiteside, A., & Vretanos, P. (2007). OpenGISő Catalogue Services Specification (No. OGC 07-006r1). Open Geospatial Consortium Inc. Retrieved from portal.opengeospatial.org/files/?artifact_id=20555

Poorazizi, M. E., Liang, S. H. L., & Hunter, A. J. S. (2012). Testing of sensor observation services: a performance evaluation. In Proceedings of the First ACM SIGSPATIAL Workshop on Sensor Web Enablement (pp. 32–38). Redondo Beach, CA, USA.: ACM. doi:10.1145/2451716.2451721

Schäffer, B. (2012). Web Processing Service Best Practices Discussion Paper. Open Geospatial Consortium.

Scholten, M., Klamma, R., & Kiehle, C. (2006). Evaluating Performance in Spatial Data Infrastructures for Geoprocessing. IEEE Internet Computing, 10(5), 34–41. doi:10.1109/MIC.2006.97

Schut, P. (2007). OpenGIS Web Processing Service (No. OGC 05-007r7). Open Geospatial Consortium Inc. Retrieved from portal.opengeospatial.org/files/?artifact_id=28772&version=2

Solntseff, N., & Yezerski, A. (1974). A survey of extensible programming languages. Annual Review in Automatic Programming, 7, Part 5, 267–307. doi:10.1016/0066-4138(74)90001-9

Steiniger, S., Poorazizi, M. E., & Hunter, A. J. S. (2013). WalkYourPlace - evaluating neighbourhood accessibility at street level. In Proceedings of the 29th Urban Data Management Symposium (Vol. XL–4/W1). London, UK: ISPRS International Archives of Photogrammetry, Remote Sensing, and Spatial Information Science.

Tamayo, A., Viciano, P., Granell, C., & Huerta, J. (2011). Empirical Study of Sensor Observation Services Server Instances. In S. Geertman, W. Reinhardt, F. Toppen, W. Cartwright, G. Gartner, L. Meng, & M. P. Peterson (Eds.), Advancing Geoinformation Science for a Changing World (Vol. 1, pp. 185–209). Springer Berlin Heidelberg.

VMware. (2006). Performance Benchmarking Guidelines for VMware Workstation 5.5. VMware, Inc.

Vretanos, P. A. (2002). OpenGIS Web Feature Service Implementation Specification. Open Geospatial Consortium Inc.

Zhu, J. (2003). Quantitative Models for Performance Evaluation and Benchmarking: Data Envelopment Analysis With Spreadsheets and Dea Excel Solver. Springer.

Appendix A

Hardware	Dell OptiPlex 960
CPU	Intel Core 2 Quad 3.0GHz
RAM	8GB
HDD	500GB
OS	Ubuntu 13.04 (64-bit)

Table 6. Experimental database server configuration

Software	**Version**
52° North	3.2.0
Deegree	3.0.4
GeoServer	2.4.3
PyWPS	3.2.1
Zoo	1.3.0
Java	Oracle JDK 7
Servlet Container	Apache Tomcat 7.0.30
Python	2.7.3
PostgreSQL/PostGIS	9.1.12/1.5.3

Table 7. Software libraries used to setup WPS servers

Appendix B

The developed Python-based geoprocessing services, WPS instances, and test scripts are publicly available at the following URLs:

Test Scripts:

- https://github.com/mepa1363/foss4g-test-script

Python-based geoprocessing services:

- https://github.com/mepa1363/wyp-server-52north-foss4g
- https://github.com/mepa1363/wyp-server-deegree-foss4g
- https://github.com/mepa1363/wyp-server-geoserver-foss4g
- https://github.com/mepa1363/wyp-server-pywps-foss4g
- https://github.com/mepa1363/wyp-server-zoo-foss4g

WPS instance:

- https://github.com/mepa1363/wyp-wrapper-52north-centralized-transit
- https://github.com/mepa1363/wyp-wrapper-deegree-centralized-transit
- https://github.com/mepa1363/wyp-wrapper-geoserver-centralized-transit
- https://github.com/mepa1363/wyp-wrapper-pywps-centralized-transit
- https://github.com/mepa1363/wyp-wrapper-zoo-centralized-transit